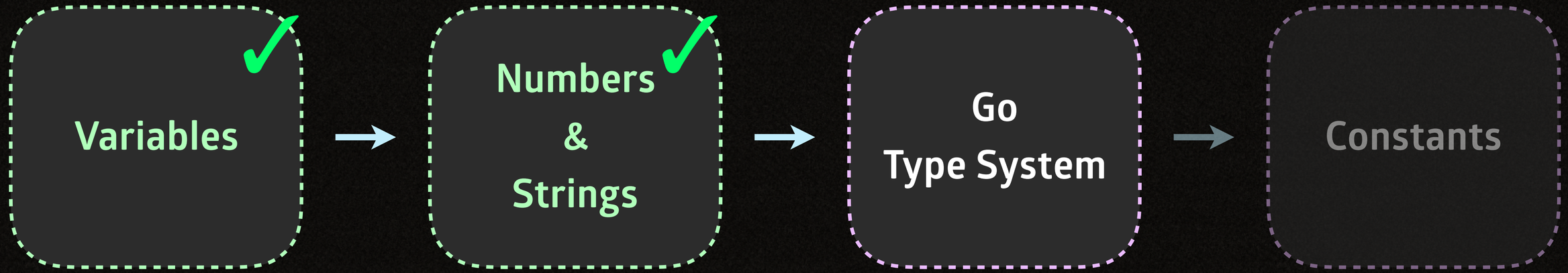
PART II

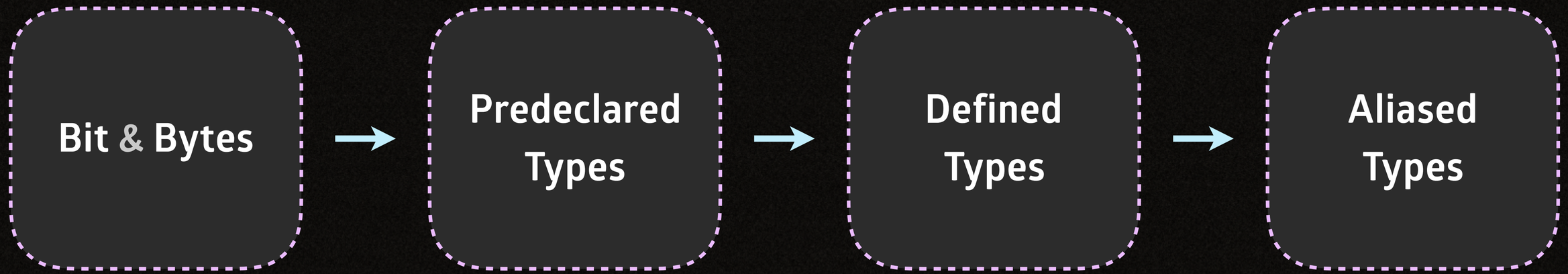
Fundamentals

Representing Data



TYPE SYSTEM

You're going to learn about the Type System of Go



Why?

The **ultimate goal** of a type system is

preventing bugs

as early as possible...

Dynamic Type System

Postpones type checking until *runtime*

Static Type System

Does type checking at *compile-time*

You still need *automated tests*
Type system can't catch logical problems

Questions.... Questions...

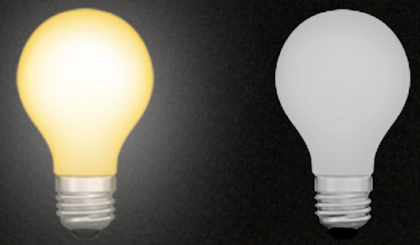
- What kinds of types are available?
- How those types interact with each other?
- What is allowed and what is disallowed?
- How to create new types?

BITS

what the heck is a bit?

BITS

a bit can be: 0 or 1



1 0

you can use bits to encode information

on off

true false

BITS

more bits = more values that you can represent

0

$$2^1 = 2$$

```
fmt.Printf("%b", 0)
```

BITS

more bits = more values that you can represent

$$1$$

$$2^1 = 2$$

```
fmt.Printf("%b", 1)
```

BITS

more bits = more values that you can represent

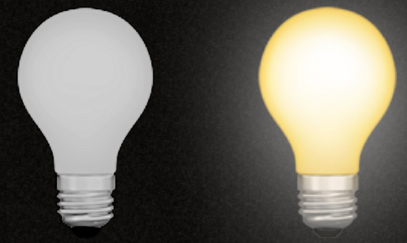
0 0

$$2^2 = 2 * 2 = 4$$

```
fmt.Printf("%02b", 0)
```

BITS

more bits = more values that you can represent



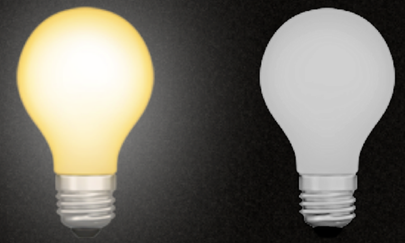
0 1

$$2^2 = 2 * 2 = 4$$

```
fmt.Printf("%02b", 1)
```

BITS

more bits = more values that you can represent



$2^1 * 1 = 2$ ← 1 0 → $2^0 * 0 = 0$

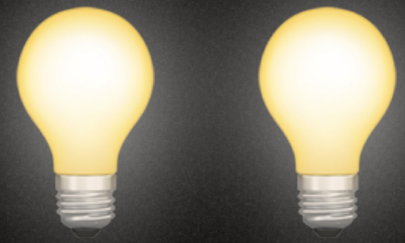
$$2^2 = 2 * 2 = 4$$

$2 + 0 = 2$

```
fmt.Printf("%02b", 2)
```

BITS

more bits = more values that you can represent



$2^1 * 1 = 2$ ← 1 1 → $2^0 * 1 = 1$

$$2^2 = 2 * 2 = 4$$

2 + 1 = 3

```
fmt.Printf("%02b", 3)
```

BITS

exercise

```
fmt.Printf("%02b", 4)
```

- Change **02b** in **Printf** to **08b** (*for example*)
- Change the **2nd parameter** to **4**
  or to a greater number
- **Analyze** how bits change

BYTES

what the heck is a byte?

BYTES

1 byte is consisting of 8 bits

byte

BYTES

1 byte is consisting of 8 bits

1 byte = 8 bits = 2^8 = 256 values

bit bit bit bit bit bit bit bit

BYTES

2 bytes equals to 16 bits

byte byte

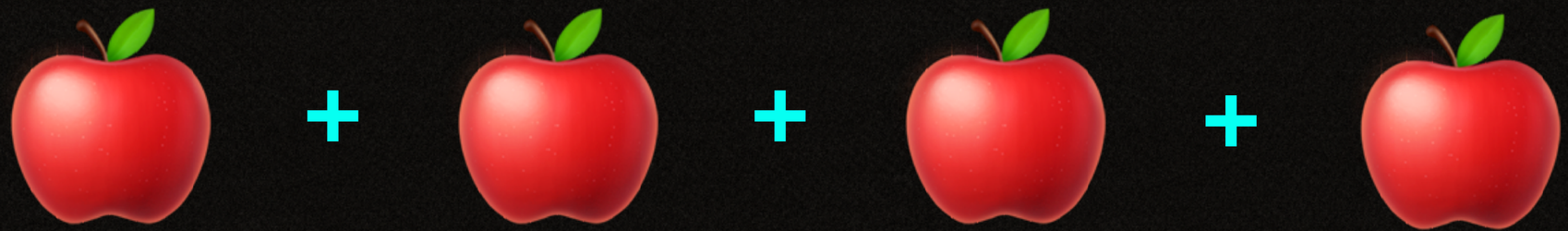
+ = 65,536 values

8 bits + 8 bits = 2^{16}

BYTES

4 bytes equals to 32 bits

byte byte byte byte



4 billion values

8 bits + 8 bits + 8 bits + 8 bits = 2^{32}

BYTES

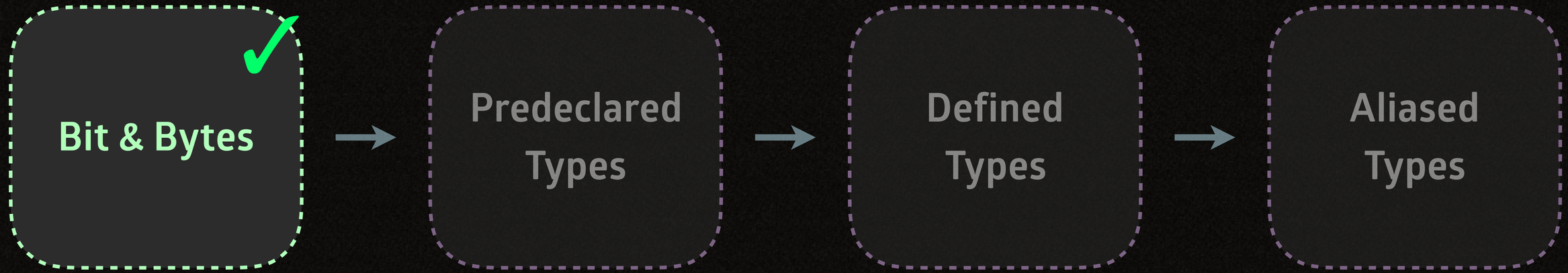
byte is an uint8 ~ It can represent 0-255 values

```go
func main() {
  var b byte

  b = 0
  fmt.Printf("%08b = %d\n", b, b)

  b = 255
  fmt.Printf("%08b = %d\n", b, b)
}
```

```
$ go run main.go
00000000 = 0
11111111 = 255
```

TYPE SYSTEM

Congrats!

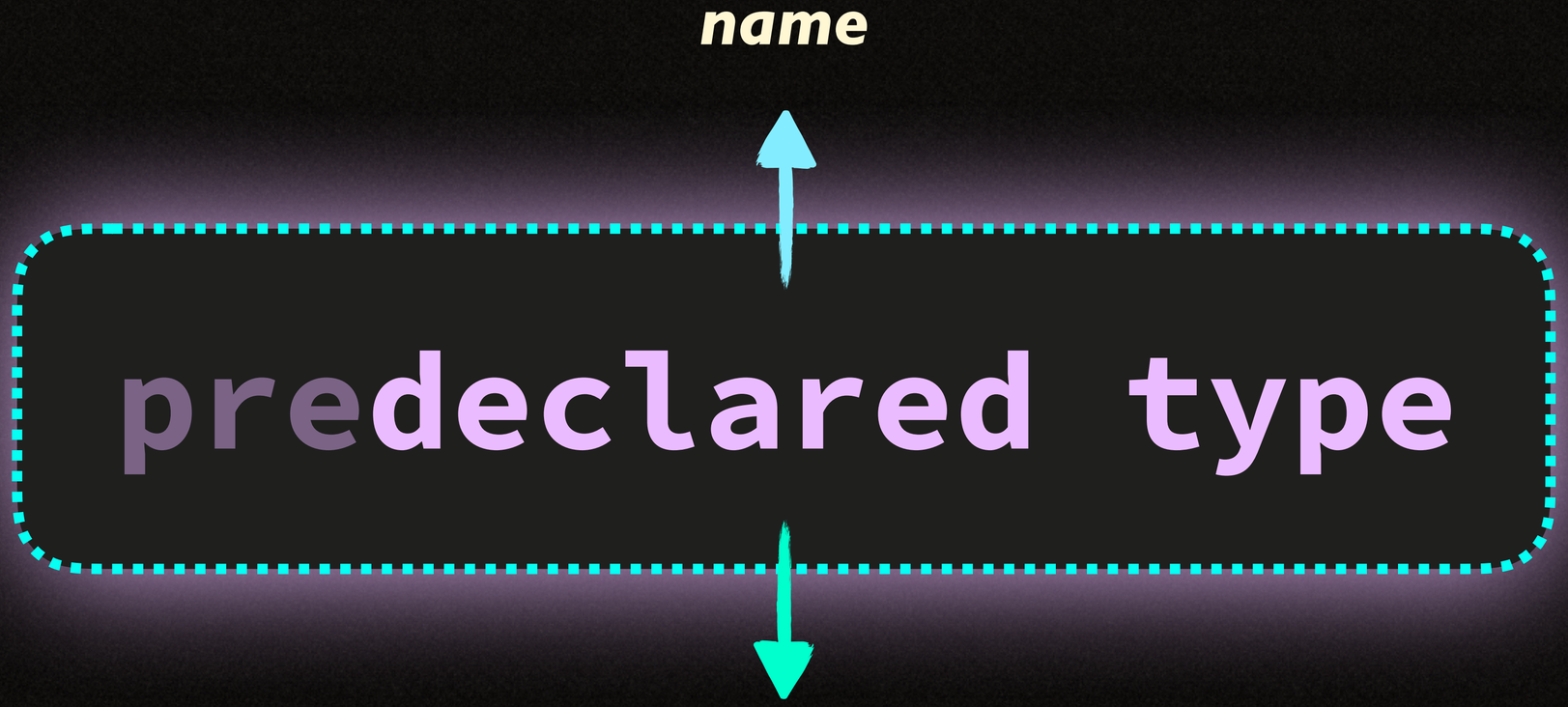


PREDECLARED TYPES

what's a predeclared type?

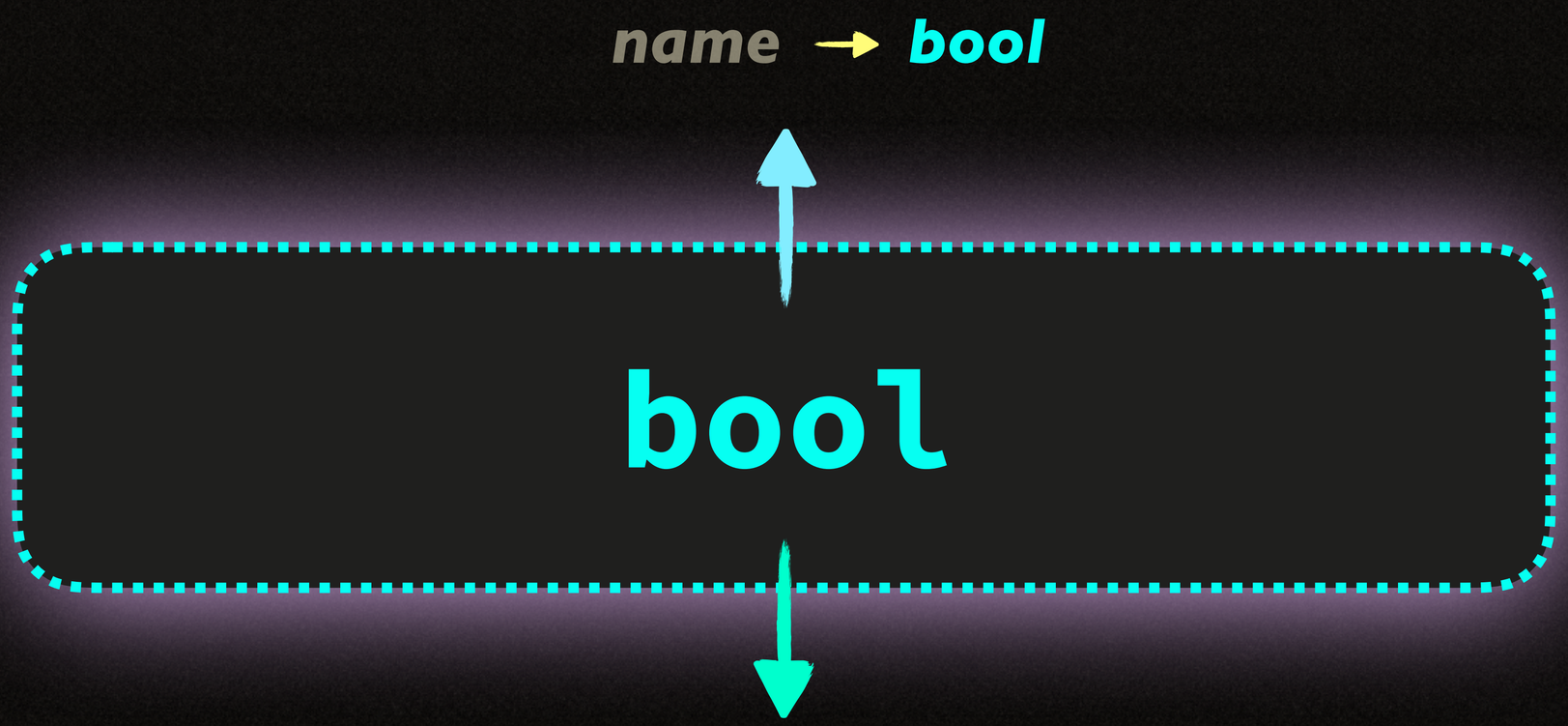
PREDECLARED TYPES

a predeclared type is a built-in type that you can use it everywhere



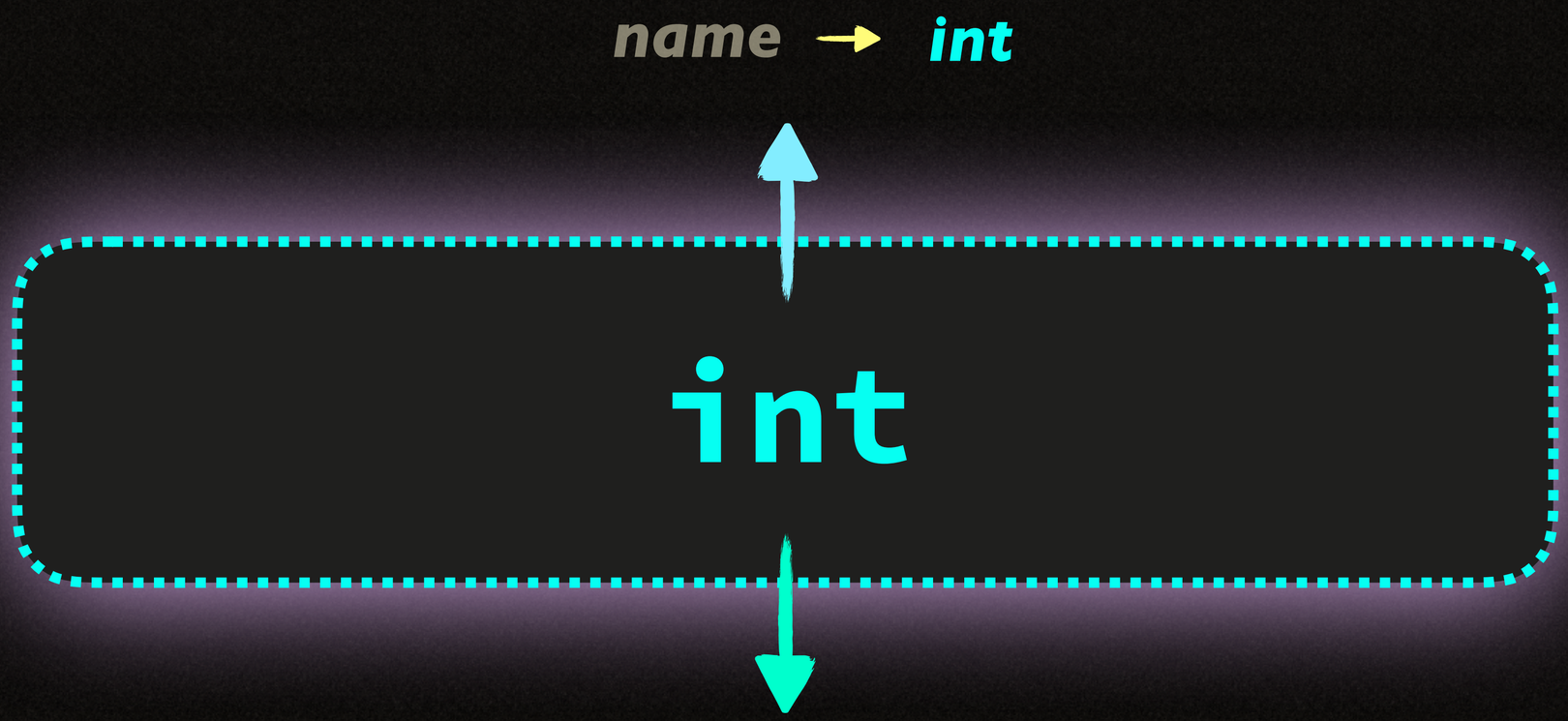
PREDECLARED TYPES

a predeclared type is a built-in type that you can use it from everywhere



PREDECLARED TYPES

a predeclared type is a built-in type that you can use it from everywhere



bool → *true or false*

string → *"hello köstebek"*
[]byte

int → like int32 or int64
int8 → -128 to 127
int16 → -32,768 to 32,767
int32 *(rune)* → -2 to +2 billions
int64 → -9 to +9 quintillions

uint → like uint32 or uint64
uint8 *(byte)* → 0 to 255
uint16 → 0 to 65,535
uint32 → 0 to 4 billions
uint64 → 0 to 18 quintillions

float32 → *3.1415927*
float64 → *3.141592653589793*

complex64 → *two float32s*
complex128 → *two float64s*

bool → *true or false*

string → *"hello köstebek"*
[]byte

int → like int32 or int64
int8 → -128 to 127
int16 → -32,768 to 32,767
int32 *(rune)* → -2 to +2 billions
int64 → -9 to +9 quintillions

uint → like uint32 or uint64
uint8 *(byte)* → 0 to 255
uint16 → 0 to 65,535
uint32 → 0 to 4 billions
uint64 → 0 to 18 quintillions

float32 → *3.1415927*
float64 → *3.141592653589793*

complex64 → *two float32s*
complex128 → *two float64s*

int8 int16 int32 int64

bit-size* = *range of values a type can represent

more bit size = more memory

int8	int16	int32	int64
-128	-32,768	-2 billions	-9 quintillions
+127	+32,767	+2 billions	+9 quintillions

bool → *true or false*

string → *"hello köstebek"*
[]byte

int → like int32 or int64

int8 → -128 to 127

int16 → -32,768 to 32,767

int32 *(rune)* → -2 to +2 billions

int64 → -9 to +9 quintillions

uint → like uint32 or uint64

uint8 *(byte)* → 0 to 255

uint16 → 0 to 65,535

uint32 → 0 to 4 billions

uint64 → 0 to 18 quintillions

float32 → *3.1415927*

float64 → *3.141592653589793*

complex64 → *two float32s*

complex128 → *two float64s*

uint8 uint16 uint32 uint64

unsigned integers can only represent positive numbers

★ **bool** → *true or false*

★ **string** → *"hello köstebek"*
[]byte

★ **int** → like int32 or int64
int8 → -128 to 127
int16 → -32,768 to 32,767
★ **int32** *(rune)* → -2 to +2 billions
int64 → -9 to +9 quintillions

uint → like uint32 or uint64
★ **uint8** *(byte)* → 0 to 255
uint16 → 0 to 65,535
uint32 → 0 to 4 billions
uint64 → 0 to 18 quintillions

float32 → *3.1415927*
★ **float64** → *3.141592653589793*

complex64 → *two float32s*
complex128 → *two float64s*

OVERFLOW

what happens when you go beyond 🚀

signed types

Min value **-** 1 → **Max Positive**

Max value **+** 1 → **Min Negative**

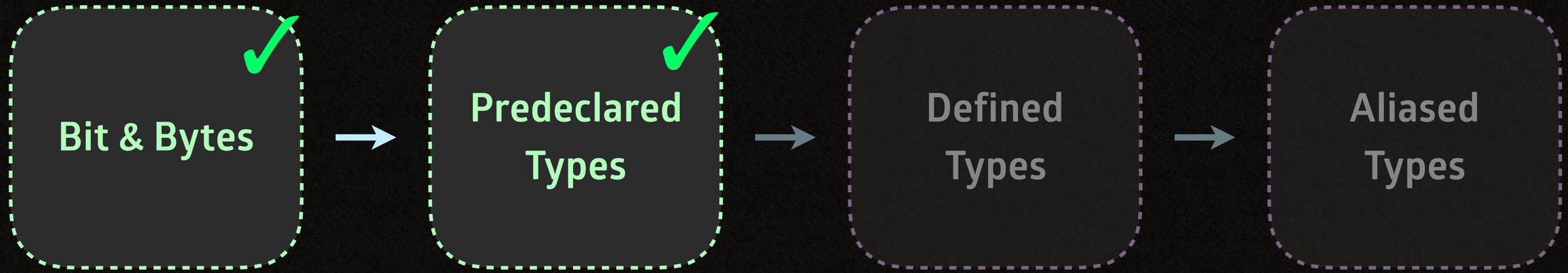
unsigned types

Min value **-** 1 → **Max Positive**

Max value **+** 1 → **Zero**

TYPE SYSTEM

Congrats!



WARNING!

🚨 This and upcoming lectures can contain **advanced** material 🚨

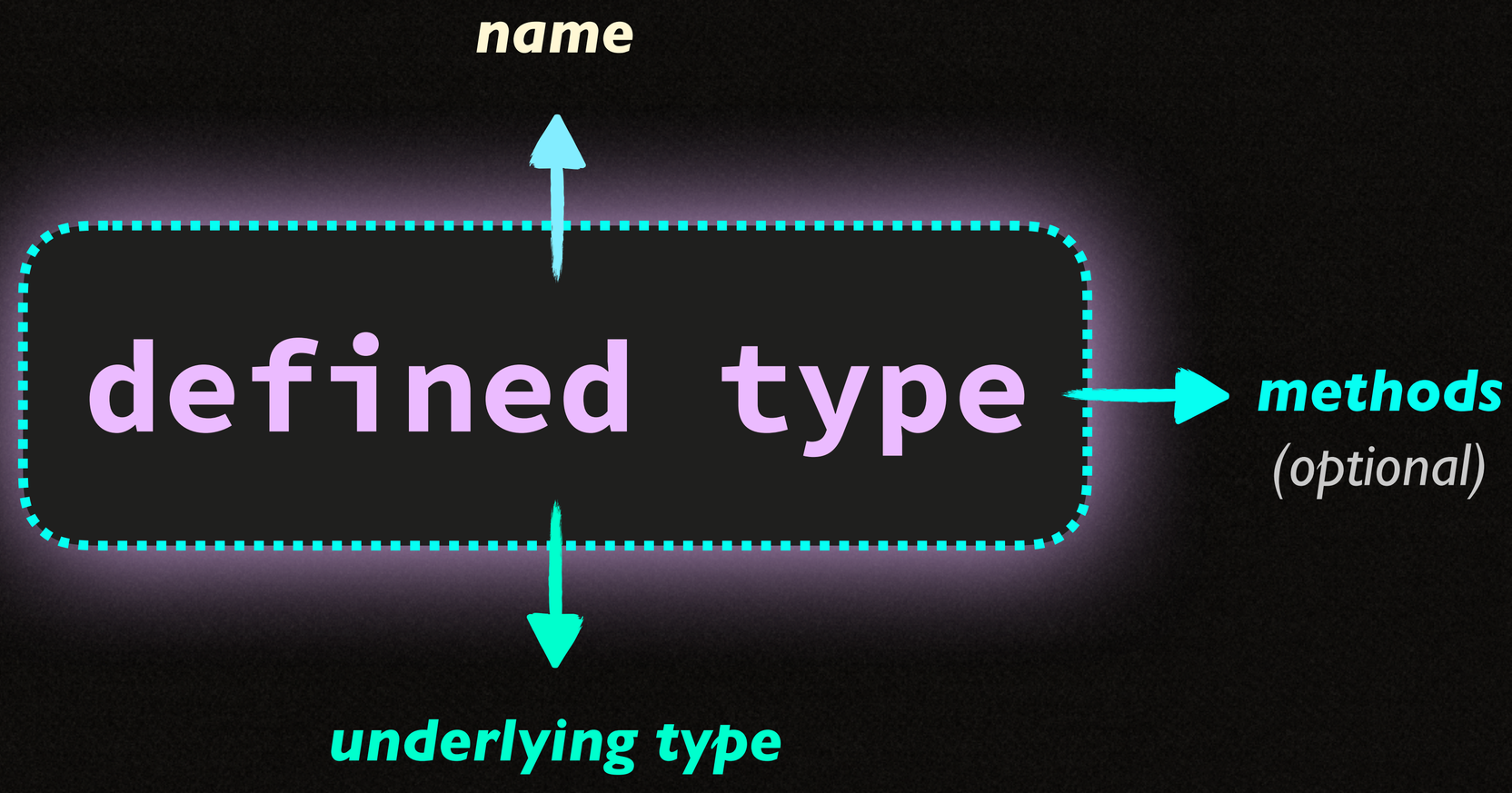
But no worries, you can always come back.

DEFINED TYPES

what's a defined type? what's an underlying type?

DEFINED TYPES

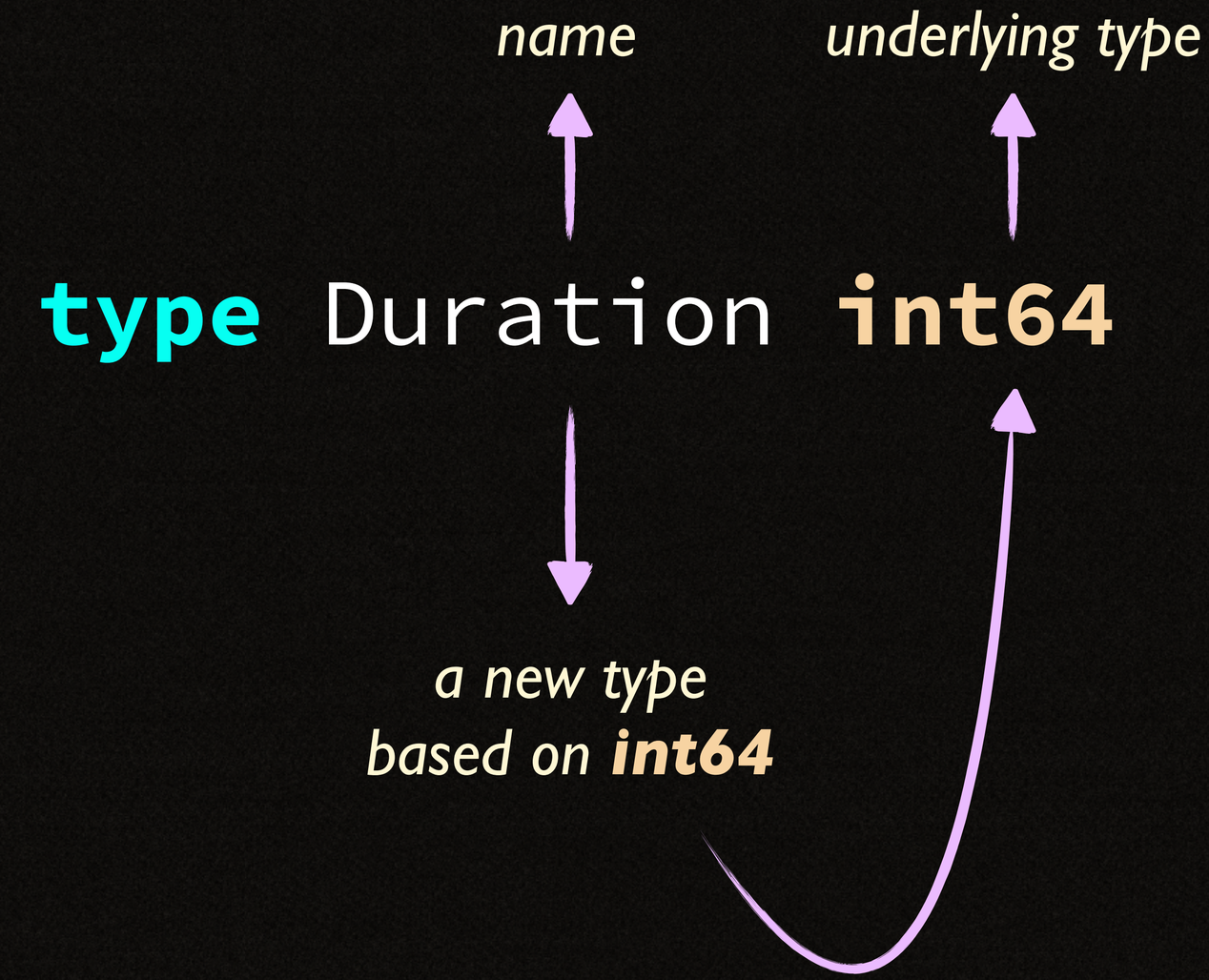
A Defined Type can only be created from another existing Type



Defined Type *is also called a* ***Named Type***

DEFINED TYPES

Duration type is created using a Type Definition



UNDERLYING TYPE

A defined type gets its properties from a type that has a real structure

```
type Duration int64
```

underlying type
int64

int64 is the real type
it has a ***structure***

★ ***operations:*** + - / * %

★ ***representation:*** -1, 0, 1, 2, ...

★ ***size***: 8 bytes

UNDERLYING TYPE

*A defined type and its underlying type are **different** types*

```
type Duration int64
```

underlying type
int64

```
var ms int64 = 1000
var ns Duration
ns = ms
```

UNDERLYING TYPE

A type can be converted to another type if they share the same underlying type and vice versa

```
type Duration int64
```

underlying type
int64

```
var ms int64 = 1000
var ns Duration

ns = Duration(ms)
ms = int64(ns)
```

✓
✓

UNDERLYING TYPE

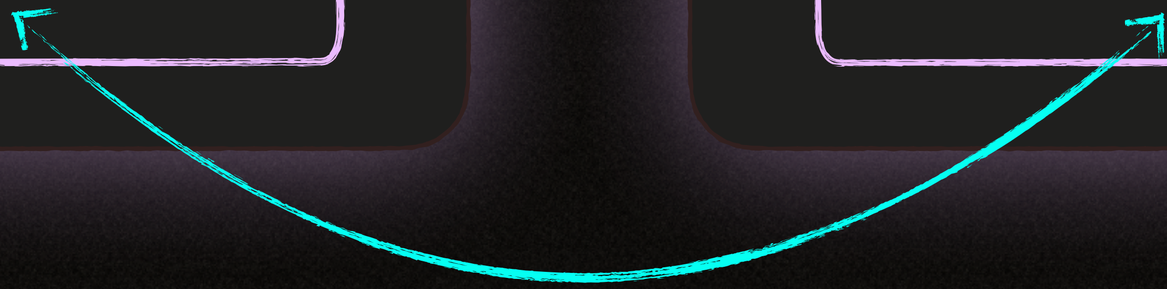
A defined type and its source type share the same underlying type

```
type Duration int64
```

underlying type
int64

```
type int64 int64
```

underlying type
int64



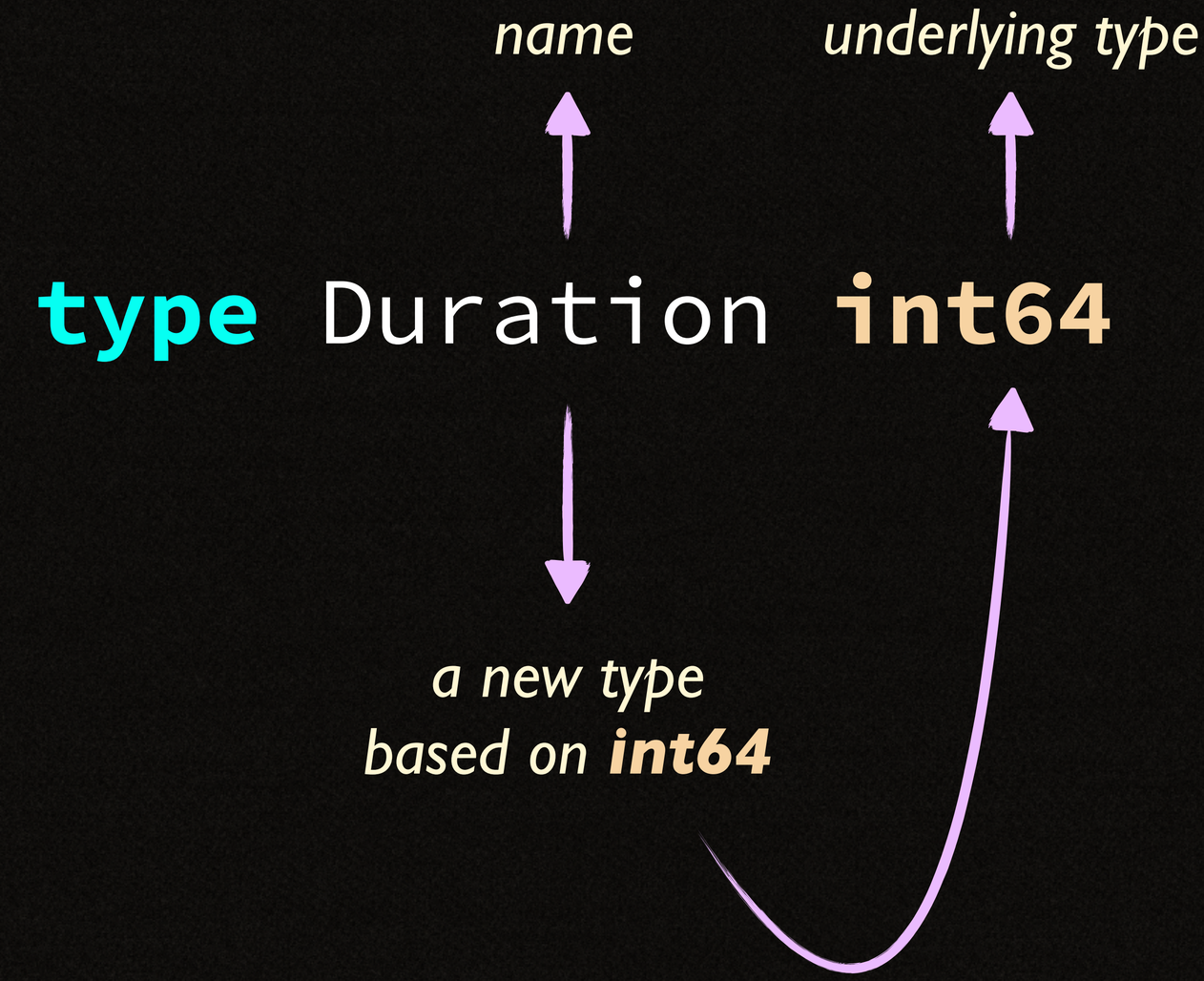
★ ***Duration*** *and* ***int64 share*** *the same* ***underlying type***

DEFINED TYPES

a real-life example

DEFINED TYPES

Duration type is defined in the time package



TYPE DEFINITION

create your own type

"If you take the trouble to name Pint and Xint,
*it's because you want them to be **distinct**.*
*This isn't C, where a **typedef** is just an **alias**."*

- Rob Pike

One of the creators of Go language

TYPE DEFINITION

You can convert value of a defined type to a type if that type's underlying type is identical

```
type gram  float64
type ounce float64

func main() {
  var g gram = 1000
  var o ounce
  o = g * 0.035274

  fmt.Printf("%g grams is %.2f ounces\n", g, o)
}
```

1 gram is 0.035274 ounce

```
ERROR: Type Mismatch
g's type is gram
o's type is ounce
```

TYPE DEFINITION

You can convert value of a defined type to a type if that type's underlying type is identical

```
type gram  float64
type ounce float64

func main() {
  var g gram = 1000
  var o ounce
  o = ounce(g) * 0.035274

  fmt.Printf("%g grams is %.2f ounces\n", g, o)
}
```

their underlying types are the same

a defined type inherits its underlying type's operations

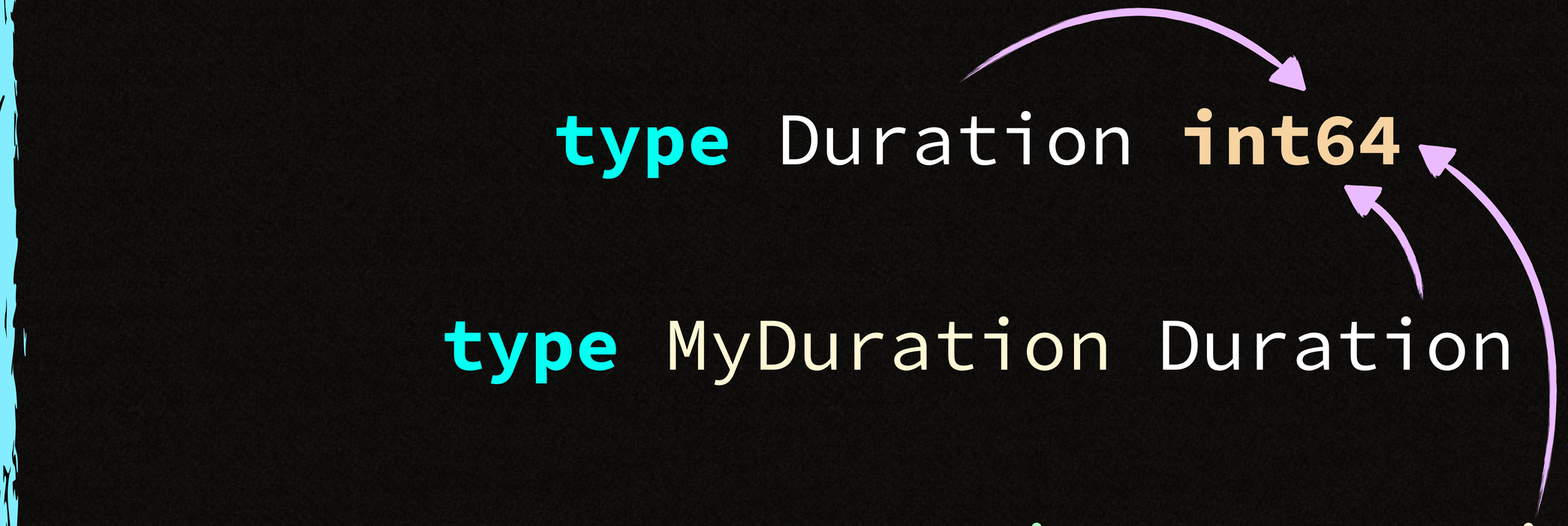
```
$ go run main.go
1000 grams is 35.27 ounce
```

UNDERLYING TYPES

Every type in Go **has** an Underlying type

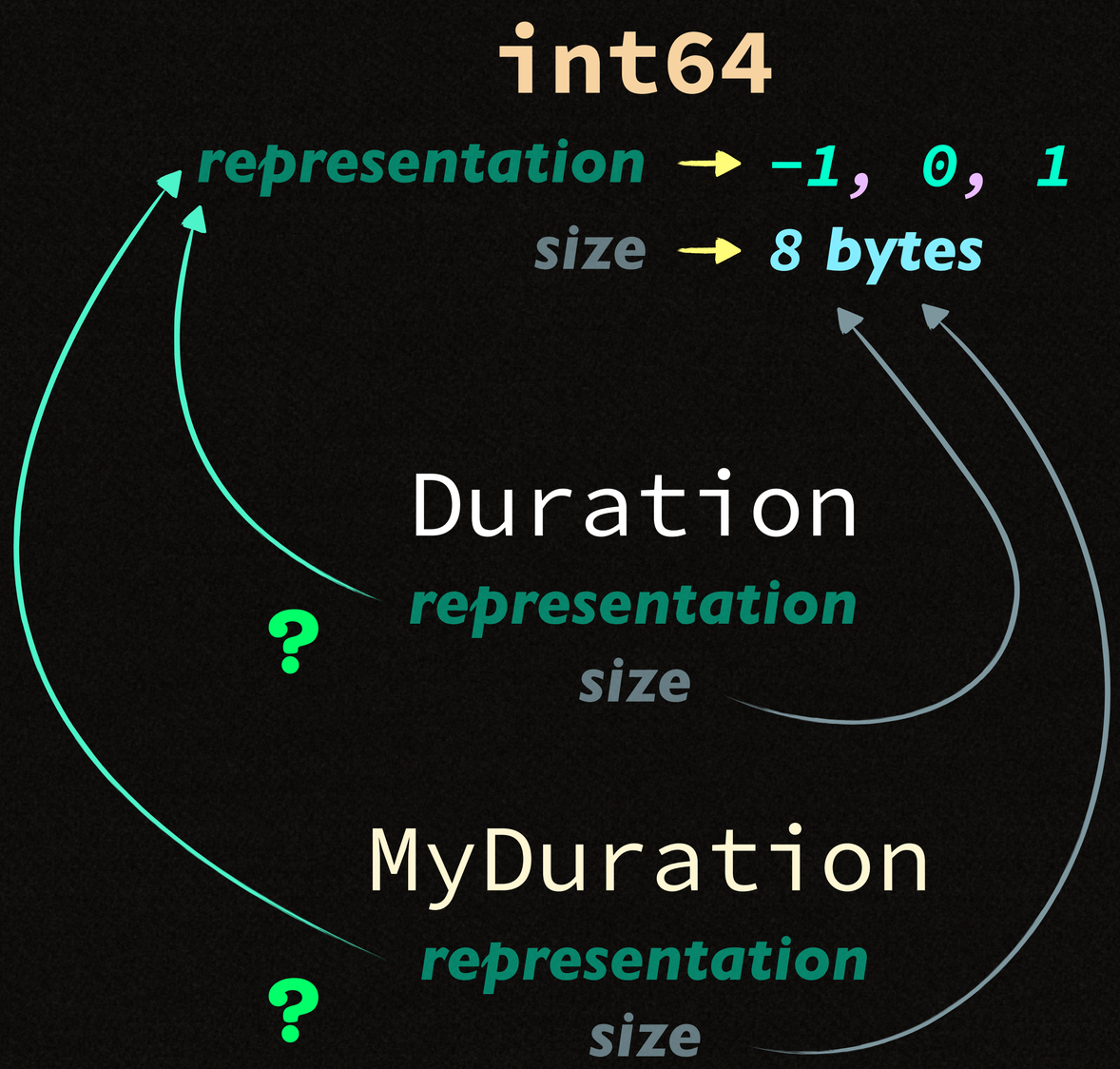
UNDERLYING TYPES

There are no type-hierarchy in Go



UNDERLYING TYPES

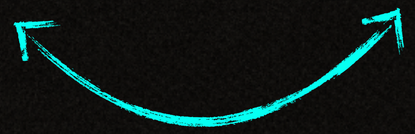
There are no type-hierarchy in Go



UNDERLYING TYPES

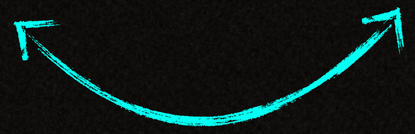
A defined type and its source type share the same underlying type

```
type int64 int64
```



Their **underlying type** is **int64**

```
type Duration int64
```

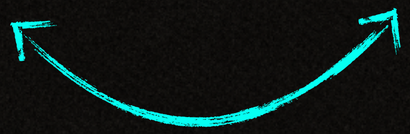


Their **underlying type** is **int64**

UNDERLYING TYPES

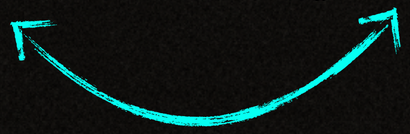
A defined type and its source type share the same underlying type

```
type MyDuration Duration
```



Their **underlying type** is **int64**

```
type MoreDuration MyDuration
```



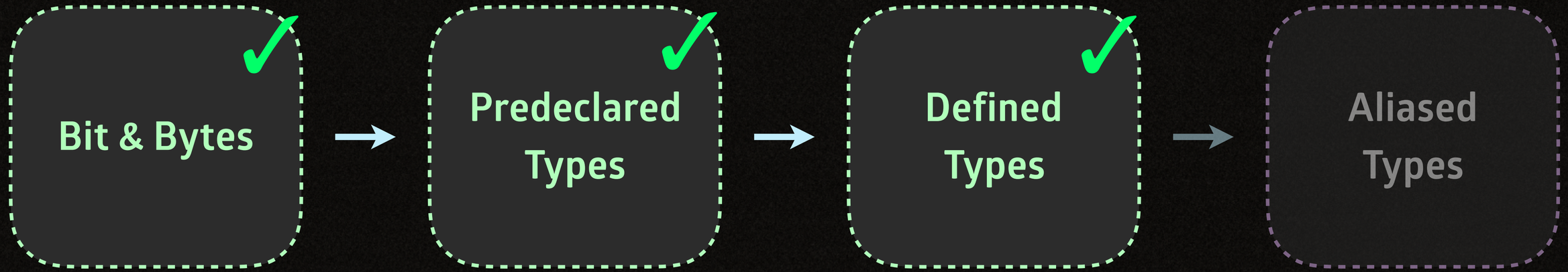
Their **underlying type** is **int64**

UNDERLYING TYPES

Every type in Go **has** an Underlying type

TYPE SYSTEM

Congrats!

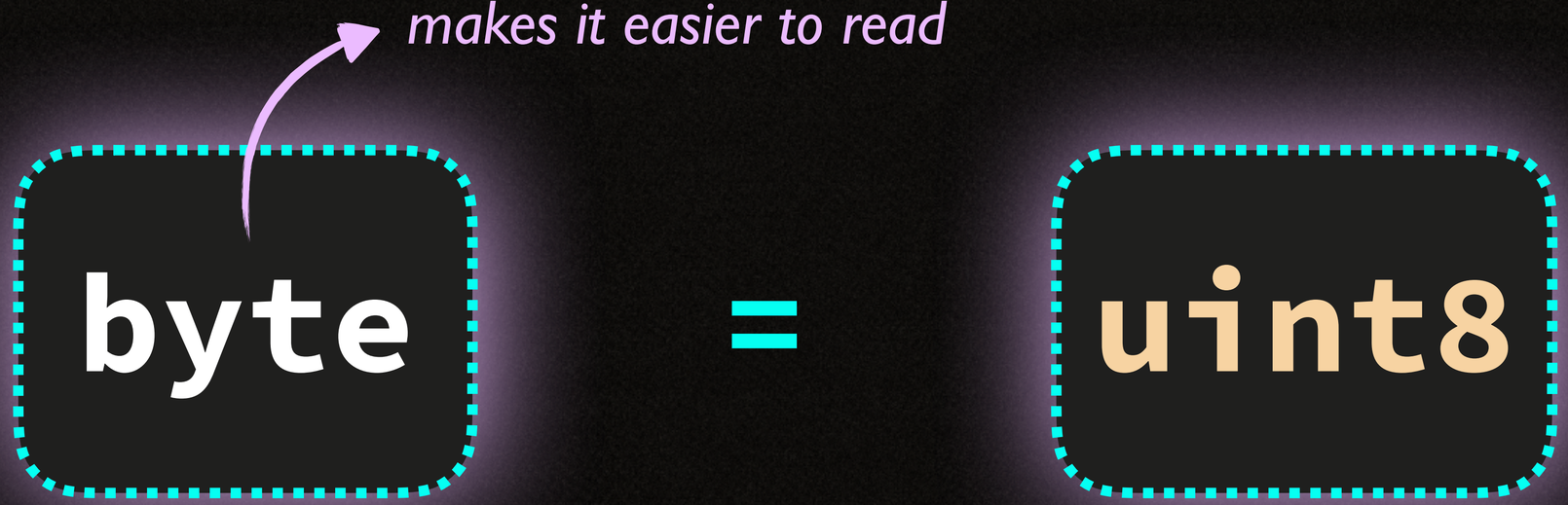


ALIASED TYPES

the very same types with a new name

ALIASED TYPES

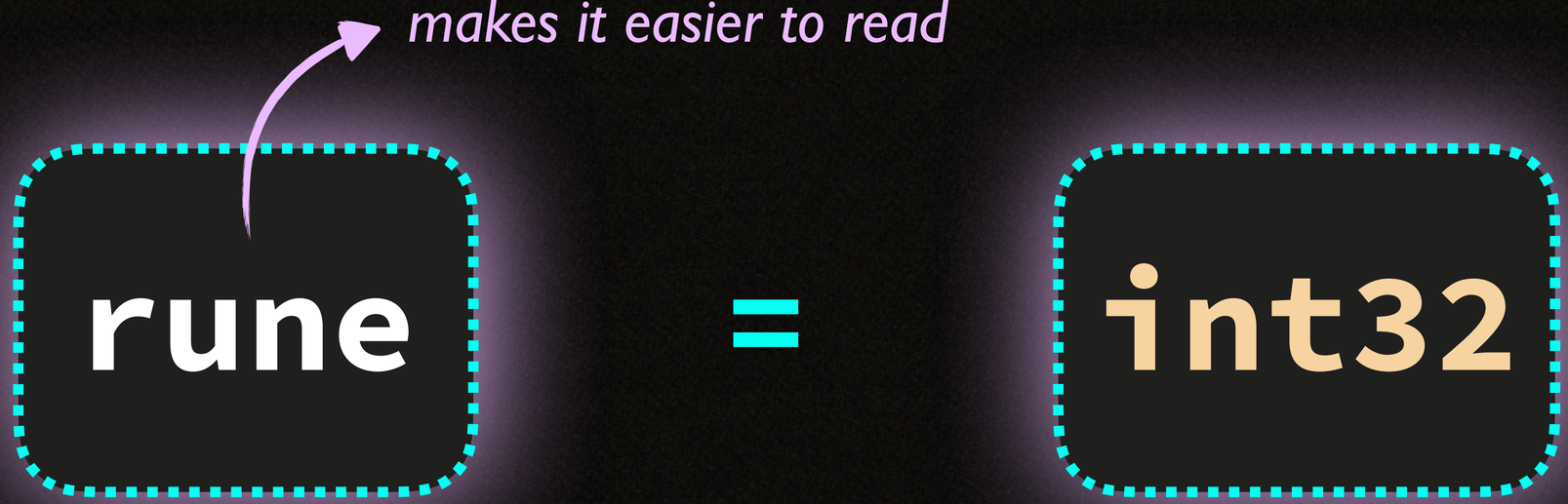
byte and uint8 are exactly the same types just with different names




```
type byte = uint8
```

ALIASED TYPES

rune and int32 are exactly the same types just with different names




```
type rune = int32
```

ALIASED TYPES

type alias declaration is not for everyday usage

```
type type = alias
```

ALIASED TYPES

example

summary

bits encode information

0 = false, off, 0

1 = true, on, 1

more bits = more information

1 bit = 0, 1

2 bits = 0, 1, 2, 3

4 bits = 0, 1, 2, 3, 4, 5, 6, 7, ..., 15

8 bits = 0, 1, 2, 3, 4, 5, 6, 7, ..., 256

16 bits = 0, ..., 65,536

32 bits = 0, ..., 4 billions

64 bits = 0, ..., 18 quintillions

byte = 8 bits

overflow and wraparound

signed types

Min value **-** 1 → Max Positive

Max value **+** 1 → Min Negative

unsigned types

Min value **-** 1 → Max Positive

Max value **+** 1 → Zero

predeclared types are built-in types

predeclared types

type	values	size
int8	-128, ..., -1, 0, 1, 127	1 byte
int16	32768, ..., -1, 0, ..., 32767	16 bytes
bool	true, false	1 byte

...

unsigned & signed

uint	int
uint8	int8
uint16	int16
uint32	int32
uint64	int64
0, 1, ...	-1, 0, 1, ...

type definition

is a declaration statement and it binds the name of the new type to a scope

```
type NewType existingType      // exported type


type newType existingType      // unexported type


func main() {
    type newType existingType
    // can only be used within the function
}
```

defined type

declares a new type using an existing type

type Duration **int64**

new type

existing type

Duration's size is 8 bytes

int64's size is 8 bytes

defined type

methods can only be attached to a defined type

```
type Duration int64

func (d Duration) Hours() float64 { ... }

func (d Duration) Minutes() float64 { ... }
```

underlying type

every type has an underlying type

type Duration **int64**

type int64 **int64**

Duration's size is 8 bytes

int64's size is 8 bytes

underlying type

a defined type **shares the same** underlying type with its source type

type Duration **int64**

type int64 **int64**

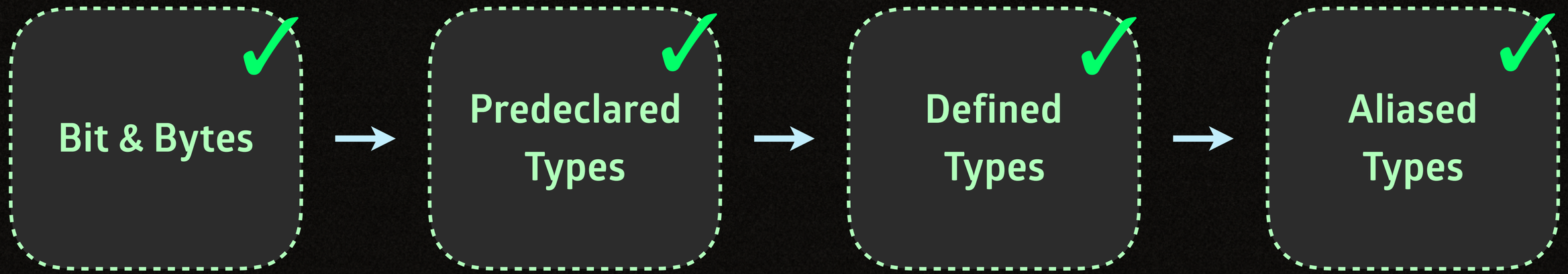
underlying type

Types with identical underlying types can be converted to each other

```
type (
    Duration        int64
    MyDuration      Duration
)

int64( MyDuration(1) )

int64( Duration(2) )


MyDuration( int64(3) )

Duration( int64(4) )
```

aliased types

```
byte    =    uint8
rune    =    int32
```

TYPE SYSTEM

Congrats!